Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования
«Южно-Уральский государственный университет»
(Национальный исследовательский университет)
Факультет вычислительной математики и информатики
Кафедра экономико-математических методов и статистики

РАБОТА ПРОВЕРЕНА
Рецензент,

_____

« » _____ 2015 г.

ДОПУСТИТЬ К ЗАЩИТЕ
Заведующий кафедрой, д. ф.-м. н., профессор
_____ А.В. Панюков

« » _____ 2015 г.

Параллельная реализация метода эллипсоидов для задач оптимизации большой размерности

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ ЮУрГУ-010400.62.2015.001.001 ВКР

Консультант,
_____

« » _____ 2015 г.

Руководитель проекта,
_____ В.А. Голодов

« » _____ 2015 г.

Автор проекта
студент группы ВМИ-413
_____ В.А. Безбородов
« » _____ 2015 г.

Нормоконтролер, к. ф.-м. н., доцент
_____ Т.А. Макаровских
« » _____ 2015 г.

Челябинск, 2015

Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования
«Южно-Уральский государственный университет»
(Национальный исследовательский университет)
Факультет вычислительной математики и информатики
Кафедра экономико-математических методов и статистики

УТВЕРЖДАЮ
Заведующий кафедрой, д. ф.-м. н., профессор
_____ А.В. Панюков
« » _____2015 г.

З А Д А Н И Е

на выпускную квалификационную работу студента

Безбородова Вячеслава Александровича

Группа ВМИ-413

1. Тема работы

   Параллельная реализация метода эллипсоидов для задач оптимизации большой размерности

   утверждена приказом по университету от « » _____2015 г. №_____

2. Срок сдачи студентом законченной работы « » _____2015 г.
3. Исходные данные к работе

   3.1. Данные из учебной литературы;

   3.2. Самостоятельно сконструированные тестовые данные.

4. Перечень вопросов, подлежащих разработке

   4.1. Изучение общей схемы работы метода эллипсоидов;

   4.2. Изучение приемов параллельной обработки данных;

   4.3. Разработка класса (типа данных) для реализации параллельно выполняемых операций над матрицами с применением библиотеки GMP;

4.4. Разработка параллельной реализации метода эллипсоидов для задачи линейного программирования;

4.5. Оценка сложности полученной реализации;

4.6. Сравнение с известными методами решения;

4.7. Тестирование;

4.8. Проверка на модельных данных.

5. Иллюстративный материал

5.1. Энергетическо-трудовой цикл 1л.

5.2. Объект, предмет и цель дипломной работы 1л.

5.3. Задачи дипломной работы 1л.

5.4. Новая концепция управления экономическими системами 1л.

5.5. Концептуальная схема модели 1л.

5.6. Общий вид модели в среде VisSim 1л.

5.7. Элементы модели энергетическо-трудового цикла 1л.

5.8. Система уравнений модели животноводства 2л.

5.9. Принцип управления 1л.

5.10. Результаты эксперимента 10л.

5.11. Заключение 1л.

5.12. Благодарность за внимание 1л.

6. Календарный план

Наименование этапов дипломной работы	Срок выполнения этапов работы	Отметка о выполнении
1. Сбор материалов и литературы по теме дипломной работы	02.02.2015 г.	
2. Исследование способов построения математической модели задачи		
3. Разработка математической модели и алгоритма		
4. Реализация разработанных алгоритмов		
5. Проведение вычислительного эксперимента		
6. Подготовка пояснительной записки дипломной работы		
Написание главы 1		
Написание главы 2		
Написание главы 3		
Написание главы 4		
Написание главы 5		
Написание главы 6		
Написание главы 7		
Написание главы 8		
Написание главы 9		
Написание главы 10		
7. Оформление пояснительной записки		
8. Получение отзыва руководителя		
9. Проверка работы руководителем, исправление замечаний		
10. Подготовка графического материала и доклада		
11. Нормоконтроль		
12. Рецензирование, представление зав. кафедрой	10.06.2015 г.	

7. Дата выдачи задания «___»_____2015 г.

Заведующий кафедрой_____/А.В. Панюков /

Руководитель работы_____/В.А. Голодов /

Студент_____/В.А. Безбородов /

АННОТАЦИЯ

Безбородов, В.А. Параллельная реализация метода эллипсоидов для задач оптимизации большой размерности / В.А. Безбородов . – Челябинск: ЮУрГУ, Факультет вычислительной математики и информатики, 2015 . – 25 с., 1 илл. Библиографический список – 15 названий.

В дипломной работе произведен анализ алгоритма метода эллипсоидов на предмет вычислительной сложности выполняемых операций. На основе результатов анализа разработана параллельная реализация метода эллипсоидов, адаптированная для решения задач оптимизации большой размерности на многопроцессорных и/или многоядерных вычислительных системах с общей разделяемой памятью.

Приведены результаты вычислительных экспериментов, продемонстрирован пример решения задачи оптимизации большой размерности с применением разработанной программной реализации.

					ЮУрГУ-010400.62.2015.001.001 ВКР			
Изм.	*Лист*	*№ докум.*	*Подп.*	*Дата*				
Разраб.		*В.А. Безбородов*			*Параллельная реализация метода эллипсоидов для задач оптимизации большой размерности : Пояснительная записка*	*Лит.*	*Лист*	*Листов*
Пров.		*В.А. Голодов*				*Д*	*4*	*25*
Рецензент						*ЮУрГУ Кафедра ЭММиС*		
Н.Контр.		*Т.А. Макаровских*						
Утв.		*А.В. Панюков*						

ОГЛАВЛЕНИЕ

Введение 7
1 Метод эллипсоидов 9
1.1 Алгоритм метода эллипсоидов 9
1.2 Анализ вычислительной сложности операций метода эллипсоидов 11
1.3 Парадигма Fork-Join 15
2 Untitled 18
2.1 Необходимость использования библиотеки GMP 18
Заключение 23
БИБЛИОГРАФИЧЕСКИЙ СПИСОК 24

Введение

Задачи оптимизации получили чрезвычайно широкое распространение в технике, экономике, управлении. Типичными областями применения теории оптимизации являются прогнозирование, планирование промышленного производства, управление материальными ресурсами, а также контроль качества выпускаемой продукции [9].

Успешность хозяйственной деятельности зависит от того, как распределяются имеющиеся ограниченные ресурсы. В связи с тем, что такая задача оптимального распределения довольно часто возникает на практике в различных сферах жизнедеятельности, актуальным становится поиск способов ускорения ее решения.

Задачи оптимизации большой размерности характеризуются высокой трудоемкостью. Использование доступного ресурса аппаратного параллелизма современных вычислительных систем рассматривается как возможность ускорения поиска их решения. Применение библиотек, реализующих поддержку арифметики произвольной точности, диктуется необходимостью достижения высокой точности при решении практических задач.

Разрабатывали, а впоследствии развивали метод эллипсоидов такие ученые, как Шор Н.З. [13], Юдин Д.Б., Немировский А.С. [15], Хачиян Л.Г. [12], Гершович В.И. [7], Стецюк П.И. [11] и др.

В работе исследован алгоритм метода эллипсоидов. Разработана его программная реализация, ориентированная на многопроцессорные и/или многоядерные вычислительные системы с общей разделяемой памятью. Показано приложение программной реализации к решению задачи оптимизации большой размерности.

Целями работы являются:

1) разработка параллельной реализации метода эллипсоидов, поддерживающей арифметику произвольной точности;
2) использование полученной реализации метода эллипсоидов для решения задачи оптимизации большой размерности.

В соответствии с поставленными целями в работе решаются следующие **задачи**:

- Исследование операций классического алгоритма метода эллипсоидов на вычислительную сложность;
- Разработка программной реализации алгоритма с распараллеливанием наиболее длительных по времени операций и поддержкой арифметики произвольной точности;
- Проверка и тестирование разработанного программного обеспечения.

Объектом исследования данной работы является метод эллипсоидов, **предметом** – параллельная реализация метода, поддерживающая арифметику произвольной точности.

Работа состоит из введения, 2 глав, заключения и списка литературы. Объем работы составляет 25 страниц. Список литературы содержит 15 наименований.

В первой главе рассматривается алгоритм метода эллипсоидов, производится его анализ на предмет вычислительной сложности с целью поиска наиболее ресурсоемких операций, нуждающихся в ускорении путем распараллеливания.

Во второй главе приведены данные, необходимые для...

В третьей главе приводится краткое описание...

В четвертой главе построена...

В пятой главе приводятся результаты вычислительных экспериментов...

В заключении перечислены основные результаты работы.

1 Метод эллипсоидов

В настоящее время большое внимание уделяется созданию автоматизированных систем планирования, проектирования и управления в различных областях промышленности. На первый план выдвигаются вопросы качества принимаемых решений, в связи с чем возрастает роль методов и алгоритмов решения оптимизационных задач в математическом обеспечении автоматизированных систем различного уровня и назначения.

Имеется несколько основных источников, порождающих задачи оптимизации: задачи математического программирования, задачи нелинейного программирования, задачи оптимального управления, задачи дискретного программирования или задачи смешанного дискретно-непрерывного типа [14].

Сфера применения методов оптимизации огромна. Создание эффективных методов оптимизации является ключом к решению многих вычислительных проблем математического программирования, особенно для задач большой размерности.

1.1 Алгоритм метода эллипсоидов

Рассмотрим алгоритм решения задачи выпуклого программирования, гарантирующий уменьшение объема области, в которой локализуется оптимум, со скоростью геометрической прогрессии, причем знаменатель этой прогрессии зависит только от размерности задачи. Этот алгоритм относится к классу алгоритмов обобщенного градиентного спуска с растяжением пространства в направлении градиента (ОГСРП) [13].

Пусть имеется задача выпуклого программирования:

$$\min f_0(x) \tag{1.1}$$

при ограничениях

$$f_i(x) \leq 0, \quad i = 1, \ldots, m, \quad x \in E_n, \tag{1.2}$$

где E_n – евклидово пространство размерности n, $f_\nu(x)$, $\nu = \overline{0,m}$ – выпуклые функции, определенные на E_n; $g_\nu(x)$ – субградиенты соответствующих функций. Пусть имеется априорная информация о том, что существует оптимальная точка $x^* \in E_n$ (не обязательно единственная), которая находится в шаре радиуса R с центром в точке x_0 (формально к системе ограничений 1.2 можно добавить ограничение $||x - x_0|| \leq R$).

Рассмотрим следующий итеративный алгоритм (при $n > 1$).

Алгоритм 1 Метод эллипсоидов

Шаг 0. Инициализация.
Положить $x_k = x_0$; $B_k = E$, где E – единичная матрица размерности $n \times n$; $h_k = \frac{R}{n+1}$ – коэффициент, отвечающий за уменьшение объема шара. Перейти к шагу 1.
Шаг 1. Вычислить

$$g(x_k) = \begin{cases} g_0(x_k), & \text{если } \max\limits_{1 \leq i \leq m} f_i(x_k) \leq 0, \\ g_{i^*}(x_k), & \text{если } \max\limits_{1 \leq i \leq m} f_i(x_k) = f_{i^*}(x_k) > 0. \end{cases}$$

Если $g(x_k) = 0$, то завершить алгоритм; x_k – оптимальная точка. Иначе перейти к шагу 2.
Шаг 2. Вычислить $\xi_k = \dfrac{B_k^T g(x_k)}{||B_k^T g(x_k)||}$. Перейти к шагу 3.
Шаг 3. Вычислить $x_{k+1} = x_k - h_k \cdot B_k \cdot \xi_k$. Перейти к шагу 4.
Шаг 4. Вычислить $B_{k+1} = B_k \cdot R_\beta(\xi_k)$, где $R_\beta(\xi_k)$ – оператор растяжения пространства в направлении ξ_k с коэффициентом β (см. определение 1.1), $\beta = \sqrt{\frac{n-1}{n+1}}$. Перейти к шагу 5.
Шаг 5. Вычислить $h_{k+1} = h_k \cdot r$, где $r = \frac{n}{\sqrt{n^2-1}}$. Перейти к шагу 1.

Определение 1.1. Оператором растяжения пространства E_n в направлении ξ с коэффициентом β называется оператор $R_\beta(\xi)$, действующий на вектор x следующим образом [14]:

$$R_\beta(\xi)x = \left(E - (\beta - 1)\xi_k \xi_k^T\right) x.$$

Рассмотрим вопрос оценки скорости сходимости метода эллипсоидов. Покажем, что данный вариант алгоритма ОГСРП сходится по функционалу со

скоростью геометрической прогрессии, причем знаменатель этой прогрессии зависит только от размерности задачи.

Лемма 1.1. Последовательность $\{x_k\}_{k=0}^{\infty}$, генерируемая алгоритмом 1, удовлетворяет неравенству

$$||A_k(x_k - x^*)|| \leq h_k \cdot (n+1), \quad A_k = B_k^{-1}, \quad k = 0, 1, 2, \ldots \tag{1.3}$$

Доказательство леммы для краткости изложения опущено и может быть найдено в [13].

Множество точек x, удовлетворяющих неравенству

$$||A_k(x_k - x)|| \leq (n+1)h_k = R \cdot \left(\frac{n}{\sqrt{n^2-1}}\right)^k,$$

представляет собой эллипсоид Φ_k, объем которого $v(\Phi_k)$ равен

$$\frac{v_0 R^n \left(\frac{n}{\sqrt{n^2-1}}\right)^{nk}}{\det A_k},$$

где v_0 – объем единичного n-мерного шара. Получаем

$$\frac{v(\Phi_{k+1})}{(\Phi_k)} = \frac{\left(\frac{n}{\sqrt{n^2-1}}\right)^n \cdot \det A_k}{\det A_{k+1}} = \frac{\left(\frac{n}{\sqrt{n^2-1}}\right)^n \cdot \det A_k}{\det R_\alpha(\xi_k) \cdot \det A_k} = \frac{1}{\alpha}\left(\frac{n}{\sqrt{n^2-1}}\right)^n =$$

$$= \sqrt{\frac{n-1}{n+1}}\left(\frac{n}{\sqrt{n^2-1}}\right)^n = q_n < 1.$$

Таким образом, объем эллипсоида, в котором локализуется оптимальная точка x^* в соответствии с неравенством (1.3), убывает со скоростью геометрической прогрессии со знаменателем q_n.

1.2 Анализ вычислительной сложности операций метода эллипсоидов

В индустрии разработки программного обеспечения известен феномен, который состоит в том, что на 20% методов программы приходится 80% вре-

мени ее выполнения [1]. Такое эмпирическое правило хорошо согласуется с более общим принципом.

Утверждение 1 (Принцип Парето). Принцип Парето, известный также как «правило 80/20», гласит, что 80% результата можно получить, приложив 20% усилий.

Относящийся не только к программированию, этот принцип очень точно характеризует оптимизацию программ [10]. В работе [3] Дональд Кнут указал, что менее 4% кода обычно соответствуют более чем 50% времени выполнения программы. Опираясь на принцип Парето, можно сформулировать последовательность действий, приводящих к ускорению работы имеющихся программ: необходимо найти в коде «горячие точки» и сосредоточиться на оптимизации наиболее трудоемких процессов.

Проанализируем подробнее вычислительную сложность операций алгоритма метода эллипсоидов. Для этого введем некоторые обозначения из области асимптотического анализа [8].

Определение 1.2. Функция $f(n)$ *ограничена сверху* функцией $g(n)$ асимптотически с точностью до постоянного множителя, т.е.

$$f(n) = O\left(g(n)\right),$$

если существуют целые N и K, такие, что $|f(n)| \leq Kg(n)$ при всех $n \geq N$.

Определение 1.3. Функция $f(n)$ *ограничена снизу* функцией $g(n)$ асимптотически с точностью до постоянного множителя, т.е.

$$f(n) = \Omega\left(g(n)\right),$$

если существуют целые N и K, такие, что $f(n) \geq Kg(n)$ при всех $n \geq N$.

Определение 1.4. Функция $f(n)$ *ограничена снизу и сверху* функцией $g(n)$ асимптотически с точностью до постоянного множителя, т.е.

$$f(n) = \Theta\left(g(n)\right),$$

если одновременно выполнены условия определений 1.2 и 1.3.

На **шаге 1** алгоритма 1 изменение текущего субградиента $g(x_k)$ происходит на основании анализа значений функций ограничений (1.2) в текущей точке x_k, т.е. анализируется последовательность значений

$$\max_{1 \le i \le m} f_i(x_k).$$

Вычислительная сложность поиска максимального из m чисел зависит от выбора алгоритма.

При использовании линейного последовательного поиска цикл выполнит m итераций. Трудоемкость каждой итерации не зависит от количества элементов, поэтому имеет сложность $T^{iter} = O(1)$. В связи с этим, верхняя оценка всего алгоритма поиска $T_m^{min} = O(m) \cdot O(1) = O(m \cdot 1) = O(m)$. Аналогично вычисляется нижняя оценка сложности, а в силу того, что она совпадает с верхней, можно утверждать $T_m^{min} = \Theta(m)$.

При использовании алгоритма бинарного поиска на каждом шаге количество рассматриваемых элементов сокращается в 2 раза. Количество элементов, среди которых может находиться искомый, на k-ом шаге определяется формулой $\frac{m}{2^k}$. В худшем случае поиск будет продолжаться, пока в массиве не останется один элемент, т.е. алгоритм имеет логарифмическую сложность: $T_m^{binSearch} = O\left(\log(m)\right)$. Резюмируем все вышесказанное относительно алгоритмов поиска в виде таблицы 1.

Таблица 1 — Оценка сложности некоторых алгоритмов поиска

Алгоритм	**Структура данных**	**Временная сложность**		**Сложность по памяти**
		В среднем	**В худшем**	**В худшем**
Линейный поиск	Массив из n элементов	$O(n)$	$O(n)$	$O(1)$
Бинарный поиск	Отсортированный массив из n элементов	$O\left(\log(n)\right)$	$O\left(\log(n)\right)$	$O(1)$

Однако при использовании некоторых алгоритмов (например, алгоритма бинарного поиска) потребуются дополнительные процедуры для упорядочива-

ния входной последовательности значений. В таблице 2 представлены асимптотические оценки наиболее известных алгоритмов сортировки массива из n элементов[1].

Таблица 2 — Оценка сложности некоторых алгоритмов сортировки[1]

Алгоритм	**Временная сложность**			**Сложность по памяти**
	В лучшем	**В среднем**	**В худшем**	**В худшем**
Быстрая сортировка	$O(n\log(n))$	$O(n\log(n))$	$O(n^2)$	$O(n)$
Сортировка слиянием	$O(n\log(n))$	$O(n\log(n))$	$O(n\log(n))$	$O(n)$
Пирамидальная сортировка	$O(n\log(n))$	$O(n\log(n))$	$O(n\log(n))$	$O(1)$
Пузырьковая сортировка	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$
Сортировка вставками	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$
Сортировка выборомм	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$
Блочная сортировка	$O(n+k)$	$O(n+k)$	$O(n^2)$	$O(nk)$
Поразрядная сортировка	$O(nk)$	$O(nk)$	$O(nk)$	$O(n+k)$

На **шаге 2** алгоритма производится вычисление нормированного обобщенного градиента

$$\xi_k = \frac{B_k^T g(x_k)}{||B_k^T g(x_k)||},$$

что подразумевает выполнение трудоемких матричных операций, таких как транспонирование, умножение на вектор и на число. Асимптотическая сложность таких операций для матрицы размерности $n \times n$ может быть оценена как $O(n^2)$.

На **шаге 3** при обновлении значения текущей точки выполняется умножение матрицы B_k на вектор ξ_k:

$$x_{k+1} = x_k - h_k \cdot B_k \cdot \xi_k.$$

Аналогично предыдущей оценке, цена такой операции составит $O(n^2)$.

[1]Более подробный анализ приведен в [6].

Шаг 4 наиболее сложен из-за необходимости вычисления оператора растяжения пространства

$$B_{k+1} = B_k \cdot R_\beta(\xi_k).$$

Если представить оператор в матричной форме (определение 1.1), то можно видеть, что в процессе его вычисления используются все вышеперечисленные операции, а также операция сложения матриц, сложность которой составляет $O(n^2)$.

На **шаге 5** осуществляется пересчет коэффициента h_k, отвечающего за уменьшение объема шара

$$h_{k+1} = h_k \cdot r.$$

Эта операция может быть выполнена за константное время, т.е. асимптотически ее сложность составит $O(1)$.

Из проведенного анализа вычислительной сложности операций, входящих в алгоритм метода эллипсоидов, можно сделать несколько выводов. Во-первых, учитывая специфику рассматриваемого класса задач (задачи оптимизации большой размерности), наиболее трудоемкие операции будут выполняться значительно дольше менее трудоемких, что приведет к сильно неравномерной загрузке вычислительной системы. Во-вторых, схема чередования сложных/простых в вычислительном смысле операций, а также выбор целевой платформы (многопроцессорные и/или многоядерные системы с общей разделяемой памятью) наталкивают на возможность использования Fork-Join Model (FJM) модели распараллеливания задач для ускорения работы алгоритма метода эллипсоидов.

1.3 Парадигма Fork-Join

В параллельном программировании, Fork-Join model (модель ветвление-объединение, FJM) – это способ запуска и выполнения параллельных участков кода, при котором выполнение ветвей завершается в специально обозначенном месте для того, чтобы в следующей точке продолжить последовательное выполнение. Параллельные участки могут разветвляться рекурсивно до тех пор, пока не будет достигнута заданная степень гранулярности задачи.

Модель была впервые сформулирована в 1963 г., и может рассматриваться как один из параллельных паттернов проектирования [4].

Различные реализации FJM обычно управляют *задачами*, *волокнами* или *легковесными нитями*, а не *процессами* уровня операционной системы, и используют *пул потоков* для их выполнения. Специальные ключевые слова позволяют программисту определять точки *возможного параллелизма*; проблема создания реальных потоков и управления ими ложится на реализацию.

В [2] приведена иллюстрация парадигмы Fork-Join. Представим ее на рисунке 1. Здесь три участка программы потенциально разрешают параллельное исполнение различных блоков. Последовательное выполнение показано сверху, в то время как его Fork-Join эквивалент снизу.

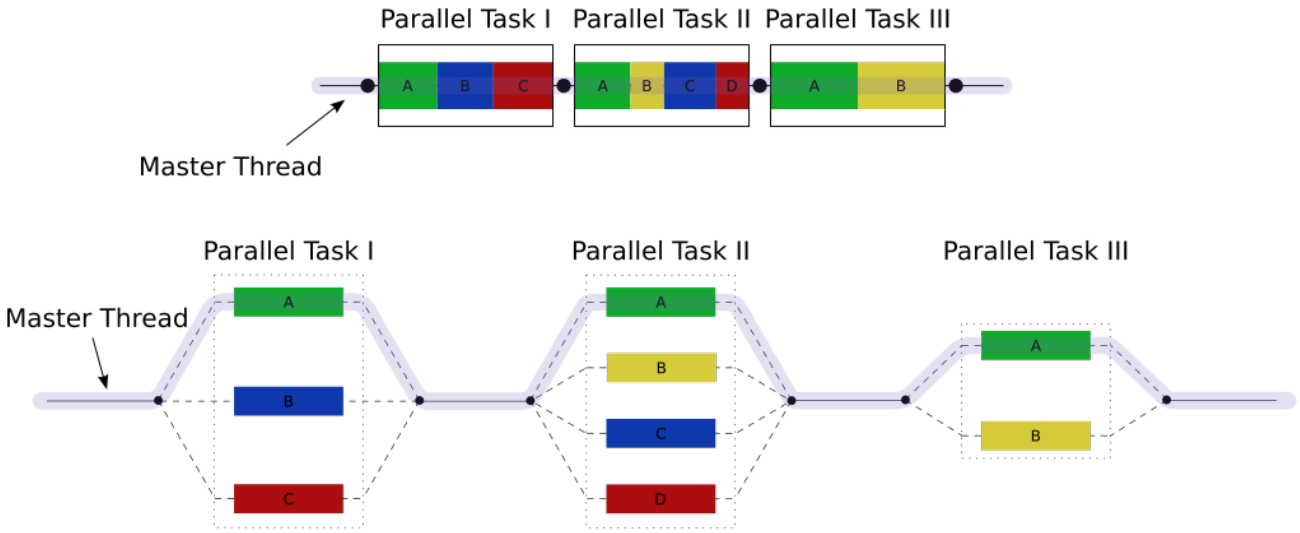

Рисунок 1 — Иллюстрация парадигмы Fork-Join

Легковесные нити, используемые в Fork-Join программировании, обычно имеют свой собственный планировщик, который управляет ими, применяя схему пула потоков. Такой планировщик может быть гораздо проще полнофункционального планировщика задач, применяемого операционной системой. Планировщики потоков общего пользования обязаны приостанавливать-/запускать потоки в обозначенных программистом местах, в то время как в парадигме Fork-Join потоки блокируются только в местах непосредственного слияния.

Модель Fork-Join – основная модель параллельного исполнения в технологии OpenMP. Также эта модель поддерживается в Java concurrency

framework, в Task Parallel Library для .NET и в Intel Threading Building Blocks. Языки программирования Cilk и Cilk Plus имеют встроенную поддержку FJM в форме ключевых слов *spawn–sync* и *cilk_spawn–cilk_sync* соответственно.

2 Untitled

2.1 Необходимость использования библиотеки GMP

Приведем численный пример, наглядно доказывающий необходимость использования библиотеки GMP для поддержки арифметики произвольной точности. Рассмотрим следующий код, представленный в листинге 1.

```
1  int main()
2  {
3    MatrixRandomFiller filler;
4    MatrixMultiplier multiplier;
5    MatrixPrinter printer( &std::cout );
6    printer.setPrecision( 20 );
7
8    std::size_t nRows = 2;
9    std::size_t nColumns = 2;
10
11   Matrix< double > matrix_a( nRows, nColumns );
12   Matrix< double > matrix_b( nRows, nColumns );
13
14   filler.fill( matrix_a );
15   filler.fill( matrix_b );
16
17   printer.print( matrix_a );
18   printer.print( matrix_b );
19
20   Matrix< double > matrix_c( nRows, nColumns );
21   multiplier.multiply( matrix_c, matrix_a, matrix_b );
22
23   Matrix< double > matrix_d( nRows, nColumns );
24   matrix_d = matrix_a * matrix_b;
25
26   printer.print( matrix_c );
27   printer.print( matrix_d );
28
29   assert( matrix_c == matrix_d );
30
31   return 0;
32 }
```

Листинг 1 — Исходный код примера

В листинге используются объекты следующих классов:

1) *Matrix< Type >* представляет в программе матрицу, элементы которой имеют тип *Type*;
2) *MatrixRandomFiller* предназначен для заполнения матрицы случайными значениями, равномерно распределенными на интервале $[0, 1)$ (в процессе заполнения используются стандартный генератор псевдослучайных чисел *std::default_random_engine* и равномерное распределение *std::uniform_real_distribution* $P(i|a, b) = \frac{1}{b-a}$ с параметрами $a = 0$, $b = 1$);
3) *MatrixMultiplier* производит перемножение матриц;
4) *MatrixPrinter* отвечает за печать матриц.

В строках 3-6 создаются и настраиваются следующие вспомогательные объекты: *filler*, *multiplier* и *printer*. *Printer* настраивается таким образом, чтобы осуществлять вывод в стандартный поток вывода *std::cout* и печатать матрицы с точностью до 20 знаков после запятой. В строках 8-9 задается размерность матриц, участвующих в данном примере. Затем происходит создание (строки 11-12), заполнение случайными значениями (строки 14-15) и печать (строки 17-18) двух операндов. В строке 21 осуществляется перемножение матриц *matrix_a* и *matrix_b* стандартным базовым алгоритмом (строка на столбец), результат которого сохраняется в объекте *matrix_c*. В строке 24 также осуществляется перемножение операндов. Отличие от строки 21 состоит в том, что в данном случае для перемножения используется функция-член класса *Matrix*. Результат второго перемножения сохраняется в переменной *matrix_d*. После этого осуществляется печать полученных результатов (строки 26-27), а также фактическая их проверка на равенство (строка 29).

Поскольку матрицы заполняются случайно, вывод программы зависит от конкретного запуска. В частности, возможен и такой вариант вывода.

$$matrix\_a = \begin{pmatrix} 0.54846871850742517918 & 0.84412510404456542190 \\ 0.95768018538410970564 & 0.80660421324152631328 \end{pmatrix},$$

$$matrix\_b = \begin{pmatrix} 0.96126112219013459814 & 0.79203852615307757112 \\ 0.79959221481843290036 & 0.90154689901948148467 \end{pmatrix},$$

$$matrix\_c = \begin{pmatrix} 1.20217751736546674124 & 1.19542672538356331557 \\ 1.56553517904925\underline{709736} & 1.48571112974158303643 \end{pmatrix},$$

$$matrix\_d = \begin{pmatrix} 1.20217751736546674124 & 1.19542672538356331557 \\ 1.56553517904925\underline{687532} & 1.48571112974158303643 \end{pmatrix}.$$

Из приведенных значений элементов матриц хорошо видно, что для одних и тех же операндов результат перемножения, полученный вне и внутри класса *Matrix*, различен, начиная с 15-го знака после запятой. При этом популярный математический сервис WolframAlpha[2] для данных операндов выдает результат

$$\begin{pmatrix} 1.2021775173654667\underline{500} & 1.1954267253835633\underline{424} \\ 1.5655351790492\underline{70011} & 1.4857111297415\underline{29851} \end{pmatrix},$$

в котором наблюдаются отличия в знаках во всех элементах матрицы. Такое расхождение неслучайно.

Если еще раз обратить внимание на листинг 1, то можно увидеть, что в численном эксперименте участвуют матрицы, все элементы которых имеют тип **double**. Согласно спецификации языка C++ [5], для представления чисел с плавающей запятой существуют несколько типов данных, характеристики которых приведены в таблице 3.

Согласно говорящим именам стандартных типов данных, а также информации из приведенной таблицы, тип **double** имеет точность, в два раза пре-

[2]http://www.wolframalpha.com

Таблица 3 — Размер и диапазон чисел с плавающей запятой в языке C++

Тип	**Размер, байт**	**Допустимый диапазон значений**
float	4	+/- 3.4e +/- 38 (точность ~7 цифр)
double	8	+/- 1.7e +/- 308 (точность ~15 цифр)
long double	8	+/- 1.7e +/- 308 (точность ~15 цифр)

вышающую **float**. В общем случае **double** имеет 15-16 десятичных знаков точности, в то время как **float** только 7.

В численном эксперименте использовался тип данных **double**, что вызвало ошибку округления примерно на 15-м знаке после запятой. Соответственно, если применять тип данных с меньшей (**float**) или большей (**long double**) точностью, меняться будет только положение десятичного знака, на котором будет начинаться расхождение, но *ошибка при этом не исчезнет*.

В такой ситуации разумным видится использование специализированных библиотек, поддерживающих более точные вычисления. Одной из таких является библиотека GMP[3] – *бесплатная (свободная)* библиотека для арифметики произвольной точности, выполняемой над знаковыми целыми, рациональными числами и числами с плавающей запятой. При этом практически не существует предела для точности вычислений, если не считать объем доступной памяти ЭВМ, на которой производятся вычисления. GMP имеет богатый набор функций, которые имеют стандартизированный интерфейс.

В основном GMP применяется в криптографических, научно-исследовательских приложениях, приложениях, отвечающих за безопасность в сети Интернет, различных системах вычислительной алгебры и т.д.

Основными целевыми платформами GMP являются Unix-подобные системы, такие как GNU/Linux, Solaris, HP-UX, Mac OS X/Darwin, BSD, AIX и проч. Также поддерживается работа на Windows в 32 и 64-битном режимах.

Библиотека GMP тщательно спроектирована для того, чтобы вычисления производились настолько быстро, насколько это возможно одновременно и для больших, и для малых операндов. Такая скорость возможна благодаря использованию машинных слов в качестве базового арифметического типа и

[3]https://gmplib.org/

быстрых алгоритмов, включающих высокооптимизированный ассемблерный код для большинства внутренних циклов для целого набора наиболее популярных современных центральных процессоров.

Первая версия GMP вышла в 1991 году. С тех пор библиотека непрерывно улучшается и поддерживается, обеспечивая выход новых версий примерно раз в год.

Начиная с версии 6, GMP распространяется одновременно под двумя лицензиями: **GNU LGPL v3** и **GNU GPL v2**. Такое лицензирование позволяет использовать библиотеку бесплатно, изменять ее и публиковать результат.

Именно свободный доступ, минимальные ограничения на использование, а также основная функция – поддержка арифметики произвольной точности – делают библиотеку GMP идеальным инструментом для использования в данной работе, особенность которой состоит в оперировании матрицами больших размерностей, что налагает дополнительные (более строгие) ограничения на точность полученного результата.

Заключение

В данной работе были приведены основные положения, выведенные из идей физической экономики. На основе этих положений была построена модель экономической системы древнего общества скотоводов-земледельцев.

В дипломной работе выполнены следующие **задачи**:

1) построена имитационная модель экономической системы древнего общества скотоводов-земледельцев;
2) проведен имитационный эксперимент;
3) проверены теоретические положения.

Можно сделать следующие выводы после проведения симуляционного эксперимента в среде VisSim:

1) модель экономической системы древнего общества скотоводов-земледельцев является жизнеспособной и правдоподобно описывает поведение древней человеческой общины.
2) модель дает возможность объяснить экономическую сущность исторических фактов относительно древних общин периода неолита;
3) проверен принцип увеличения доли свободного времени в общем фонде социального времени по ходу развития общины.

Данная модель может быть улучшена путем более точного описания различных хозяйственных процессов, происходивших в экономике общины. Также возможно применение тензорной методологии для описания этих хозяйственных процессов, при этом уравнения примут более понятный внешний вид, не утратив своего содержания.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Boehm, B. W. Industrial software metrics top 10 list / Barry W. Boehm // IEEE Software 4. — 1987. — no. 9 (September). — 84–85.
2. Fork–join model . — URL: http://en.wikipedia.org/wiki/Fork-join_model (дата обращения: 12.04.2015).
3. Knuth, D. An empirical study of fortran programs / Donald Knuth // Software – Practice and Experience. — 1971. — 105–33.
4. McCool, M. Structured Parallel Programming: Patterns for Efficient Computation / Michael McCool, James Reinders, Arch Robinson. — MK, 2013.
5. Stroustrup, B. The C++ Programming Language / Bjarne Stroustrup. — Addison-Wesley, 2013.
6. Вирт, Н. Алгоритмы и структуры данных. Новая версия для Оберона + CD / Пер. с англ. Ткачев Ф. В. / Н. Вирт. — М.: ДМК Пресс, 2010. — 272 с.: ил.
7. Гершович, В. И. Метод эллипсоидов, его обобщения и приложения / В. И. Гершович, Н. З. Шор // Кибернетика. — 1982. — №5.
8. Грин, Д. Математические методы анализа алгоритмов / Д. Грин, Д. Кнут. — М.: Мир, 1987. — 120 с., ил.
9. Данилин, А. И. Основы теории оптимизации (постановки задач) [Электронный ресурс] : электрон. учеб. пособие / А. И. Данилин. — Минобрнауки России, Самар. гос. аэрокосм. ун-т им. С.П. Королева (нац. исслед. ун-т). – Электрон. текстовые и граф. дан. (1,2 МБайт). – Самара, 2011. — 1 эл. опт. диск (CD-ROM).
10. Макконнелл, С. Совершенный код. Мастер-класс / Пер. с англ. / С. Макконнелл. — М.: Издательство «Русская редакция», 2010. — 896 стр.: ил.
11. Стецюк, П. И. Методы эллипсоидов и r-алгоритмы / П. И. Стецюк. — Нац. акад. наук Украины, Ин-т кибернетики им. В. М. Глушкова, Акад. транспорта, информатики и коммуникаций. – Кишинэу: Эврика, 2014. — 488 с.

12. Хачиян, Л. Г. Полиномиальные алгоритмы в линейном программировании / Л. Г. Хачиян // Ж. вычисл. матем. и матем. физ. — 1980. — С. 51–68.
13. Шор, Н. З. Метод отсечения с растяжением пространства для решения задач выпуклого программирования / Н. З. Шор // Кибернетика. — 1977. — № 1. – С. 94–95.
14. Шор, Н. З. Методы минимизации недифференцируемых функций и их приложения / Н. З. Шор. — Киев: Наук. думка, 1979. — 200 с.
15. Юдин, Д. Б. Информационная сложность и эффективные методы решения выпуклых экстремальных задач / Д. Б. Юдин, А. С. Немировский // Экономика и мат. методы. — 1976. — т. 12, вып. 2. – С. 357–369.